Dedicated Software Analysis Tools

Nicolas Anquetil, Stéphane Ducasse, Muhammad Usman Bhatti

Dedicated Software Analysis Tools

by Nicolas Anquetil, Stéphane Ducasse and Usman Bhatti

The data and software analysis platform Moose allows for the quick development of dedicated tools that can be customized at different levels. These tools are crucial for large software systems that are subject to continuous evolution.

The lifetime of large systems (such as those that support the activities of banks, hospitals, insurance companies and the army) can be measured in decades. Such software systems have become a crucial component for running the day-to-day affairs of our society. Since these systems model important aspects of human activity, they must undergo continuous evolution that follows the evolution of our society. For example, new laws, economical constraints or requirements force large software systems to evolve. Previous studies have shown that undertaking this evolution can represent up to 90% of total software effort [1]. Controlling such systems and ensuring they can evolve is a key challenge: it calls for a detailed understanding of the system, as well as its strengths and weaknesses. Deloittes recently identified this issue as an emerging challenge [2].

From an analysis of the current situation, four key facts emerge.

1. Despite the importance of software evolution for our economy, it is not considered to be a relevant problem (and in fact, is considered a topic of the past): for example, currently there are no EU research axes that focus on this crucial point while buzzwords such as "big data" and "the Cloud" attract all the attention.
2. People seem to believe that the issues associated with software analysis and evolution have been solved, but the reality is little has been accomplished.
3. New development techniques such as Agile Development, Test Driven Development, Service-Oriented Architecture and Software Product Lines cannot solve the problems that have accumulated over years of maintenance on legacy systems and it is impossible to dream of redeveloping even a small fraction of the enormous quantity of software that currently exists today.
4. Software evolution is universal: it happens to any successful software, even in projects written with the latest and coolest technologies. The productivity increases that have been achieved with more recent technologies will further complicate the issue as engineers produce more complex code that will also have to be maintained. There are tools that propose some basic analyses in terms of the "technical debt" (i.e., that put a monetary value on bad code quality), however, knowing that you have a debt does not help you take action to improve code quality.

Typical software quality solutions that assess the value of some generic metrics at a point in time are not adapted to the needs of the developers. Over the years we have developed Moose, a data and software analysis platform. We have previously presented Moose [3], but in this article, we want to discuss some of the aspects we learnt while selling tools built on top of Moose. In conjunction with the clients of our associated company Synectique, we identified that an adequate analysis infrastructure requires the following elements.

The first is *dedicated processes and tools* which are needed to approach the specific problems a company or system might face. Frequently software systems use proprietary organization schemes to complete tasks, for example, to implement a specific bus communication between components. In such cases, generic solutions are mostly useless as they only give information in terms of the "normal", low-level concepts available in the programming language used. Large software systems need to be analyzed at a higher abstraction level (e.g., component, feature or sub-system). This supports reverse engineering efforts. In Moose, we offer a meta model-based solution where the imported data is stored independently of the programming language. This approach can be extended to support proprietary concepts or idioms, and new data can be supported by merely adapting the model and defining the proper importer. Once the information is imported, analysts can take advantage of the different tools for crafting software analyses that are tailored to meet their needs.

The second element is *tagging*. End users and/or reengineers often require a way to annotate and query entities with expert knowledge or the results of an analysis. To respond to this need, end users and reengineers are provided with a tagging mechanism which allows them to identify interesting entities or their groups. An interesting case which highlights the use of this mechanism is the extraction of a functional architecture from structural model information. Once experts or analyses have tagged entities, new tools and analyses (such as a rule-based validation) use it (by querying) to advanced knowledge and create more results.

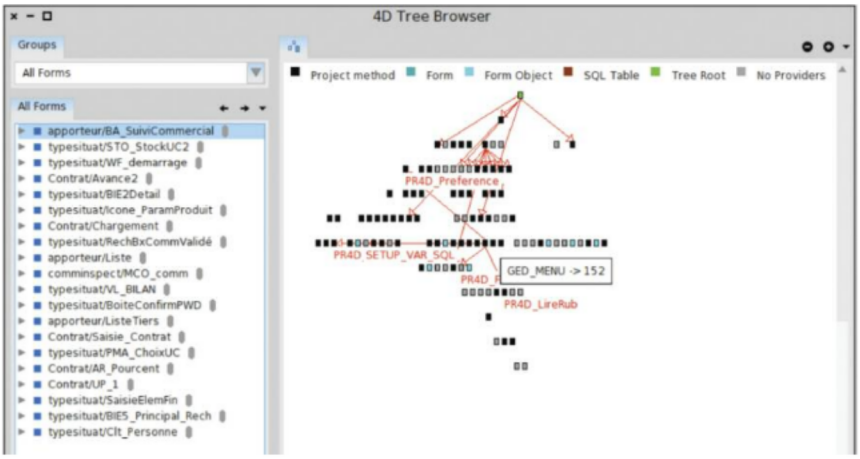

Figure 1: A dependency analyzer for legacy code.

The third element is the *dependency nightmare* analysis and remediation tool. Large and/or old software systems have often suffered from architectural drift for so long that there is little or no architecture left. All parts of a software system are intrinsically linked and frequently, loading three modules can mean loading the complete system. The challenge is how to deal with this fine grained information at a large grain level. We propose an advanced cyclic dependency analysis and removal tools as well as a drill-down on architectural views. Figure 1 shows the tool revealing recursive dependencies for impact analysis.

The fourth element is a *trend analysis* solution. Instead of a punctual picture of a system's software state, it is desirable to understand the evolution of the quality of entities. As the source code (and thus the software entities) typically evolve in integrated development environments, independently from the dedicated, off-the-shelf, software quality tools, computing quality analysis trends requires changes (e.g., add, remove, move or rename) of individual software entities identified. We propose a tool that computes such changes and the metric evolutions. Figure 2 shows the changes computed on two versions (green: entity added, red: entity removed) and the evolution of quality metrics for a change. Queries may be expressed in terms of the changes (i.e., "all added methods") or in terms of the metric variations (i.e., "increase of CyclomaticComplexity > 5").

Figure 2: Trends analysis.

Conclusion

Software evolution and maintenance is, and will continue to be, a challenge for the future. This is not because a lack of research advances but rather because more and more software is being created and that software is destined to last longer. In addition, any successful software systems must evolve to adapt to global changes. Our experience shows that while problems may look similar on the surface, key problems often require dedicated attention (e.g., processing, analyses and tools). There is a need for dedicated tools that can be customized at different levels, such as models, offered analyses and the level of granularity.

Links:

http:// www.moosetechnology.org
http://www.synectique.eu

References:

[1] C. Jones: "The Economics of Software Maintenance in the Twenty First Century", 2006, http://www.compaid.com/caiinternet/ezine/capersjones-maintenance.pdf

[2] B. Briggs et al.: "Tech Trends 2014, Inspiring Disruption", White Paper, Deloitte University Press, 2014, http://www.deloitte.com/assets/Dcom-Luxembourg/Local%20Assets/Documents/Whitepapers/2014/dtt_en_wp_techtrends_10022014.pdf

[3] O. Nierstrasz, S. Ducasse, T. Gîrba: "The story of Moose: an agile reengineering environment", ESEC/SIGSOFT FSE 2005: 1-10.

Please contact:

Stéphane Ducasse
Inria, France
E-mail: stephane.ducasse@inria.fr

Mining Open Software Repositories

by Jesús Alonso Abad, Carlos López Nozal and Jesús M. Maudes Raedo

With the boom in data mining which has occurred in recent years and higher processing powers, software repository mining now represents a promising tool for developing better software. Open software repositories, with their availability and wide spectrum of data attributes are an exciting testing ground for software repository mining and quality assessment research. In this project, the aim was to achieve improvements in software development processes in relation to change control, release planning, test recording, code review and project planning processes.

In recent years, scientists and engineers have started turning their heads towards the field of software repository mining. The ability to not only examine static snapshots of software but also the way they have evolved over time is opening up new and exciting lines of research towards the goal of enhancing the quality assessment process. Descriptive statistics (e.g., mean, median, mode, quartiles of the data-set, variance and standard deviation) are not enough to generalize specific behaviours such as how prone a file is to change [1]. Data mining analysis (e.g., clustering, regression, etc.) which are based on the newly accessible information from software repositories (e.g., contributors, commits, code frequency, active issues and active pull requests) must be developed with the aim of proactively improving software quality, not only reactively responding to issues. Open source software repositories like Sourceforge and GitHub provide a rich and varied source of data to mine. Their open nature welcomes contributors with very different skill sets and experience levels and the absence or low levels of standardized workflow enforcement make them reflect 'close-to-extreme' cases (as opposed to the more structured